let:

ES6 新增了let命令，用来声明变量。它的用法类似于var，但是所声明的变量，只在let命令所在的代码块内有效（针对var的）能用let就不要var

1、变量名不能重复；同一个作用域不可使用 let 重复声明同一个变量

2、声明的全局变量不会追加到window（窗口）对象身上；

```
var a = 10;
var a = 20;
    console.log(window.a);//20
//ES6  let声明的变量不能重复
let user = 'ABC';
    console.log(user);//ABC

//let声明的变量不会追加到window对象身上
let num = 100;
    console.log(num);//100
    console.log(window.num);//undefined
```

3、声明的变量不存在变量提升；

```
 //let声明的变量不存在变量提升
   console.log(a);//Uncaught ReferenceError: Cannot access 'a' before initialization
   let a = 100;
```

4、存在块级作用域：代码块；声明的变量仅在块级作用域内有效

```
for(let i=0; i < 5; i++){
            setTimeout(function(){
                console.log(i);
            }, 0);
        }//0 1 2 3 4
//先执行同步，在执行异步
        let j = 1;
        {
            //块级作用域
            let j = 2;
```

```
            console.log(j);//2
        }
        console.log(j);//1
```

5、暂时性死区：

ES6 规定，如果区块中存在let和const命令，这个区块对这些命令声明的变量，从一开始就形成了封闭作用域；也就是声明的变量就“绑定”（binding）这个区域，不再受外部的影响

```
//暂时性死区
        for (let n = 0; n < 5; n++) {
            console.log(n);
            let n = 20;
        }
```

const：

静态变量（常量，但是并不是常量），能用const不要用let； “值”不可以修改，本质：引用地址不可变；const的作用域与let命令相同：只在声明所在的块级作用域内有效

特点：

- 不允许重复声明；
- 存在块级作用域；
- 声明必须赋初始值；
- 值不允许修改；
- const在声明变量的时候是需要赋值的；

```
//使用const声明静态变量（常量）
const NUM = 100;
//NUM = 200;
    console.log(NUM);

//使用const声明一个静态变量
```

```
const PERSON = {
      username: '小米',
      age: 19
   };
     PERSON.gender = 'female';
     //PERSON = {a:1};//引用地址直接发生了改变
     console.log(PERSON);
     //{username: '小米', age: 19, gender: 'female'}

        //真正的静态变量（常量）
        const Obj = Object.freeze({
            a: 1,
            b: 2
        });
        Obj.c = 3;
        console.log(Obj);//{a: 1, b: 2}

        //服务器响应数据
        const data = {
            code: 1,
            data: [],
            msg: '操作成功'
        };
        //进行数据库查询，把查询到的数据放到data.data里面
        //return data;
```

2、const 用于声明一个或多个常量，声明时必须进行初始化，且初始化后值不可再修改：

```
const PI = 3.1415926;
      PI = 3.14;
      PI = PI + 10;
```

3、const 的本质: const 定义的变量并非常量，并非不可变，它定义了一个常量引用一个值。

```
//使用 const 定义的对象或者数组，其实是可变的。
        // 创建常量对象
        const car = {
            type: "Fiat",
```

```
            model: "500",
            color: "white"
        };
        // 修改属性:
        car.color = "red";
        // 添加属性
        car.owner = "Johnson";
        console.log(car.owner); //Johnson

//但是我们不能对常量对象重新赋值:
        const car = {
            type: "Fiat",
            model: "500",
            color: "white"
        };
        car = {
            type: "Volvo",
            model: "EX60",
            color: "red"
        };
        console.log(car);

//以下实例修改常量数组:
        // 创建常量数组
        const cars = ["Saab", "Volvo", "BMW"];
        // 修改元素
        cars[0] = "Toyota";
        // 添加元素
        cars.push("Audi");
        console.log(cars); // ['Toyota', 'Volvo', 'BMW', 'Audi']

        //但是我们不能对常量数组重新赋值:
        const car1 = ["Saab", "Volvo", "BMW"];
        car1 = ["Toyota", "Volvo", "Audi"]; // 错误
        console.log(car1);
```

所谓的静态变量，是指 引用地址 不可改变;

- 基础变量：栈内存，栈内存里面的数据不可修改，如果要修改一个变量的值，需要创建一个新的内存空间，变量指向这个新的内存空间；const声明的变量的引用地址不可变，给我们的感觉就是值不可改变；

- 引用类型：堆内存，堆内存里面的数据是可以修改的；只要不修改引用地址，就没问题；

解析:const:静态变量（常量，但是并不是常量），能用const,就不用let

- (1)“值”不可以修改，本质:引用地址不可变
- (2)所谓的静态变量，是指引用地址不可修改
- (3）基础类型:栈内存，栈内存里面的数据不可修改，如果要修改一个变量的值，需要重新创建一个新的内存空间，变量指向这个新的内存空间，const.声明的变量的引用地址不可变。
- (4）引用类型:堆内存，堆内存里面的数据是可以修改的，只要不修改引用地址就可以。
- (5) const,在声明变量的时候是需要赋值的;

浏览器的顶层对象window和nodejs的顶层对象global合而为 globalThis

```
//globalThis  是顶层对象
//浏览器下的顶层对象是    window
//nodejs里面的顶层对象是  global
//浏览器的顶层对象window和nodejs的顶层对象global合而为 globalThis
   console.log(window);
   console.log(globalThis);
```

解构赋值:

按照某种数据的结构进行解析数据然后赋值给变量 解析这个结构:

- 数组：有顺序要求;

```
//数组的解构赋值
let [a, b, c] = [1, 2, 3];
console.log(a, b, c);

let [i, j] = [100, [300]];
console.log(i, j);

//n m 交换值  使用解构赋值
let n = 300, m = 400;

/**
常规的变量的值交换
let tmp = n;
n = m;
 m = tmp;
*/
```

```
[n, m] = [m, n];
console.log(n, m);

//作为函数的参数
function sum([x, y]){
    return x + y;
    }
console.log(sum([100, 200]));
```

- 对象：没有顺序要求，用起来很方便，可以设置默认值；

```
//对象的解构赋值
let { a, b, c } = { a: 200, b: 300, c:100 };
console.log(c);//100

let { max, min } = Math;
console.log(max(100, 800, 600));//800
console.log(min(100, 800, 600));//100
```

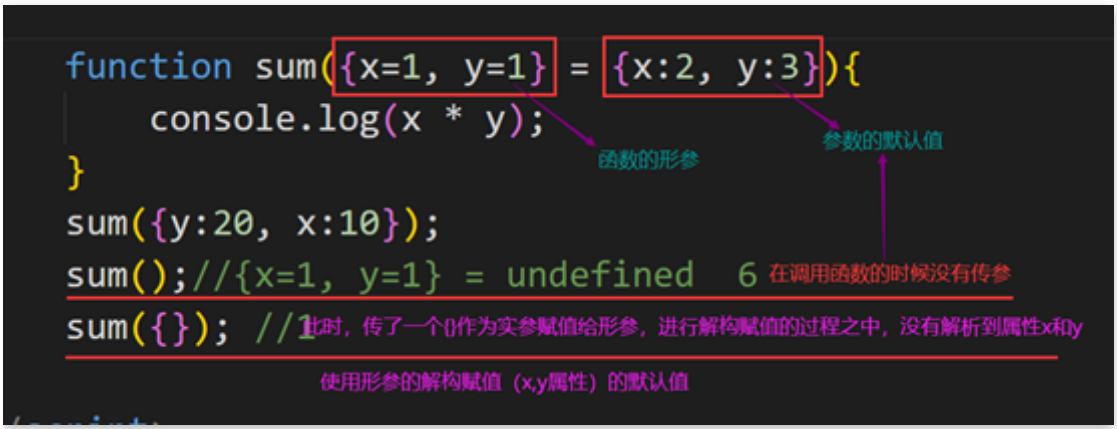
默认值：只有解析出来的值是undefined才会使用默认值；

```
//对象的解构赋值  支持设置默认值
//默认值：undefined的情况下才会使用默认值
let { a, b, c = 500 } = { c:null, a: 200, b: 300 };
console.log(a, b, c);//200 300 null

let { a, b, c = 500 } = { c:undefined, a: 200, b: 300 };
console.log(a, b, c);//200 300 500

//解构赋值在函数里面的使用
function mysqlConnect({host='127.0.0.1', user, password, port=3306}){
   console.log(host, user, password, port);
   }
    mysqlConnect({
        user:'root',
        password:'root',
        host:'192.168.22.34'
    });

```

```
    function sum({x=1, y=1} = {x:2, y:3}){
        console.log(x * y);
    }
        sum({y:20, x:10});
        sum();//{x=1, y=1} = undefined  6
        sum({}); //1
        sum({y:null}); //0
        sum({x:true}); //1
```

- 字符串解构赋值

```
//字符串解构赋值
let [a, b, c] = 'hello';
    console.log(a);//h
    console.log(b);//e
    console.log(c);//l
```



模板字符串

模板字符串（template string）是增强版的字符串，用反引号（`）标识，特点：

- 字符串中可以出现换行符
- 可以使用 ${xxx} 形式输出变量

```
//变量拼接
let name = 'shansan';
let result = `欢迎${name}到来`;
console.log(result);
```

简化对象写法

ES6 允许在大括号里面，直接写入变量和函数，作为对象的属性和方法，这样的书写更加简洁。

```
let name = "张三";
let age = 18;
let speak = function () {
    console.log(this.name);
};

//属性和方法简写
let person = {
    name,
    age,
    speak
};

console.log(person.name);
console.log(person.age);
person.speak();
```

变量的作用域：

全局变量、局部变量、块级作用域

ES6this指向：this 指向详细解析（箭头函数） - Chris-dc - 博客园 (cnblogs.com)